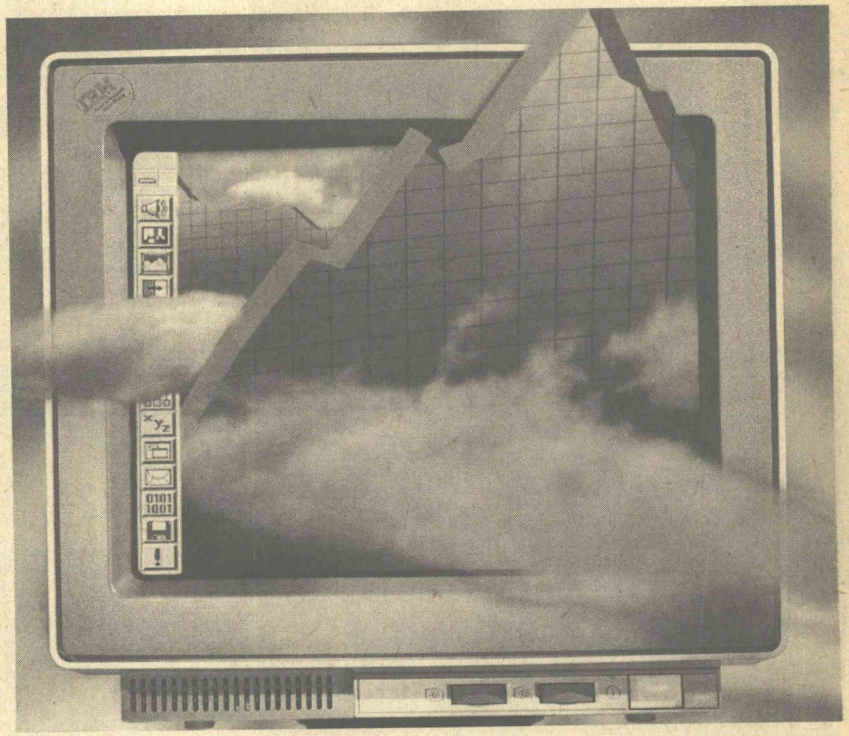
ลออ เพิ่มสมบัติ

การเขียน กราฟิก อย่างมีระบบ ตอน การเข้าถึงหน่วยความจำโดยตรง (จบ)

เป็นตอนจบของการเข้าถึงหน่วยความจำโดยตรง โดยให้แนวคิดในนำเวิร์ดโปรเซสซิงมาใช้งาน

เมื่อคุณต่อจอวิดีโอกับเครื่องพีซีคุณจำเป็นต้องติดตั้งในคอมพิวเตอร์ในส่วนของบอร์ดวงจรที่เรียกว่าตัวปรับกราฟิก บอร์ดนี้จะมีวงจรที่จำเป็นที่จะเปลี่ยนข่าวสารจากคอมพิวเตอร์เข้าสู่การแสดงผล ในส่วนของฮาร์ดแวร์จะประกอบด้วยอะไรบ้างและจะเข้าถึงฮาร์ดแวร์จากโปรแกรมได้อย่างไร เป็นสิ่งที่น่าจะกล่าวถึงเพื่อเสริมมโนภาพให้แก่โปรแกรมเมอร์รุ่นเยาว์ ผมจึงขอสาธยายให้คุณได้ทราบต่อไปนี้

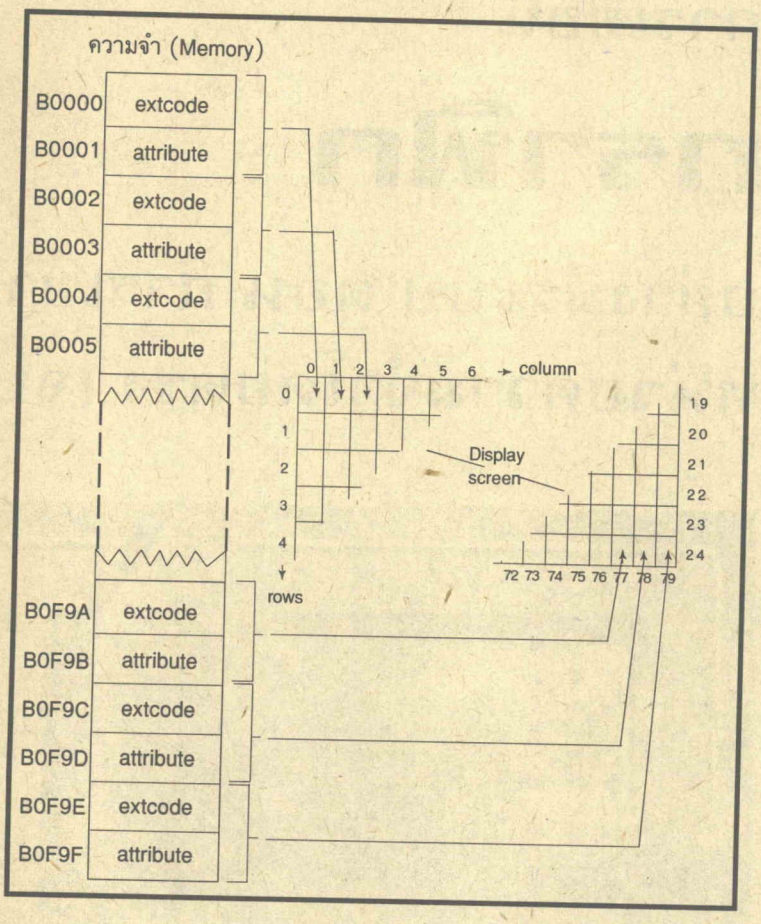
การสื่อสารระหว่างตัวปรับและคอมพิวเตอร์เริ่มด้วยการใช้ส่วนของหน่วยความจำที่มีอยู่ ส่วนหน่วยความจำส่วนนี้ตามลักษณะทางกายภาพจะอยู่ที่บอร์ดตัวปรับและถูกเข้าถึงได้โดยไมโครโปรเซสเซอร์ และส่วนที่ส่งผลออกทางจอภาพ ไมโครโปรเซสเซอร์จะสอดแทรกค่าต่าง ๆ เข้าไปในหน่วยความจำและอ่านค่าเช่นเดียวกับที่ทำได้โดยแรม (RAM) ส่วนฮาร์ดแวร์ที่ต้องนำไปแสดงทางจอภาพจะเอาค่าเหล่านี้มาจากหน่วยความจำ แล้วส่งตัวอักษรที่สมนัยกันออกจอภาพ



ตัวอักษรที่นำออกแสดงจะมีทั้งหมด 25 บรรทัด ในแต่ละบรรทัดมี 80 ตัวอักษร รวมแล้วจะมีตัวอักษรทั้งหมด 2,000 ตัวอักษร ตัวอักษรแต่ละตัวก็จะเกี่ยวข้องกับแอดเดรส โดยเฉพาะในหน่วยความจำที่ต้องการแสดง โดยตัวอักษรแต่ละตัวใช้ 2 ไบต์ ไบต์แรกจะเกี่ยวข้องกับรหัสแอสกี (ASCII) ซึ่งมีค่าตั้งแต่ 0 ถึง 255 (0 ถึง ff ในเลขฐานสิบหก) และอีก 1 ไบต์จะเก็บค่าคุณลักษณะจำเพาะ (attribute) นั่นคือเราต้องใช้ถึง 4,000 ไบต์ในหน่วยความจำเพื่อแสดงอักษร 2,000 ตัวออกจอภาพ

หน่วยความจำจะใช้กับจอโมโนโครม (Monochrome) เริ่มที่แอดเดรส B0000 (hex) ไปจนถึง B0F9F (9F9 เลขฐานสิบหกจะตรงกับ 3999 ในเลขฐานสิบ) รูปที่ 1 แสดงถึงความสัมพันธ์ระหว่างหน่วยความจำกับส่วนที่แสดงออกทางจอภาพ

รูปที่ 1 แสดงถึงจอโมโนโครมกับการเข้าถึงหน่วยความจำ



ถ้าคุณมีจอสีต่อกับเครื่องคอมพิวเตอร์ และต้องการแสดงในรูปเท็กซ์ หน่วยความจำที่จะแสดงเท็กซ์จะมีตำแหน่งของแอดเดรสต่าง ๆ กัน โดยเริ่มที่ B8000 และเพิ่มขึ้นเรื่อย ๆ จนถึง B8F9F นั่นคือคุณจะแปลงแอดเดรส B0000 ในเท็กซ์ให้เป็นแอดเดรส B8000

เมื่อคุณต้องการเรียกรูทีนที่มีอยู่ในไลบรารีของ C เพื่อแสดงตัวอักษรออกทางจอภาพก็ต้องเรียกรูทีนในรอมไบออสจะนำเอาตัวอักษรออกทางจอภาพด้วยการแทนค่าสำหรับตัวอักษรในส่วนที่เป็นรหัสแอสกี ส่วนขยาย (Extended Code) และส่วนที่เป็นลักษณะเฉพาะ (attribute) ในแอดเดรสในหน่วยความจำของตัวปรับ แต่เราอาจใช้วิธีลัด เนื่องจากถ้าโปรแกรมใส่ค่าไบต์เหล่านี้โดยตรงในหน่วยความจำจะช่วยประหยัดเวลามาก เพราะว่าเรากำจัดการเรียกซ้ำได้

พอยน์เตอร์ Far

เมื่อเราทราบแอดเดรส เราจะใส่ค่าไปในหน่วยความจำโดยอาศัยพอยน์เตอร์ คือจะต้องวางตัวอักษรลงในโลเคชันแรกของหน่วยความจำที่จะใช้กับจอภาพ และจะใช้ประโยคข้างล่างนี้เข้าช่วยประกาศให้พอยน์เตอร์ชี้ไปที่จำนวนเต็ม (pointer-to-int) ซึ่งทำให้สามารถจะใช้ทางเลือกได้ 2 ทางคือรหัสแอสกีขยายและคุณลักษณะจำเพาะ จึงต้องกำหนดให้พอยน์เตอร์เท่ากับแอดเดรสของหน่วยความจำที่จะออกจอภาพ ในที่สุดเราก็เข้าถึงโลเคชันนี้โดยใช้โอเปอเรเตอร์ทางอ้อม

```
int*ptr; /*looks plausible*/
— —
ptr = 0xB0000; /*but*/
— —
*(ptr) = ch; /*won't work*/
```

แต่โชคไม่ดี เพราะเราได้ทำผิดเสียแล้ว เนื่องจากตัวแปรพอยน์เตอร์ประกอบด้วยปริมาณ 2 ไบต์ ขณะที่แอดเดรส B0000 (hex) ประกอบด้วย 2 ไบต์กับ 1 ไบต์ครึ่ง (5-digit) ซึ่งไม่สามารถทำให้เป็น 2 ไบต์ได้เลย

เซกเมนต์ (Segment)

พอยน์เตอร์ปกติใช้การเก็บแอดเดรสใน 1 เซกเมนต์เท่านั้น ไมโครโปรเซสเซอร์ตระกูล 80x86 (8088, 8086, 80286, 80386, 80486) เซกเมนต์เป็นส่วนของหน่วยความจำที่มี 10,000 (hex) หรือ 65,536 (ในเลขฐานสิบ) ไบต์ ภายในเซกเมนต์แอดเดรสจะมีค่าเริ่มตั้งแต่ 0 ถึง FFFF (ในเลขฐานสิบหก)

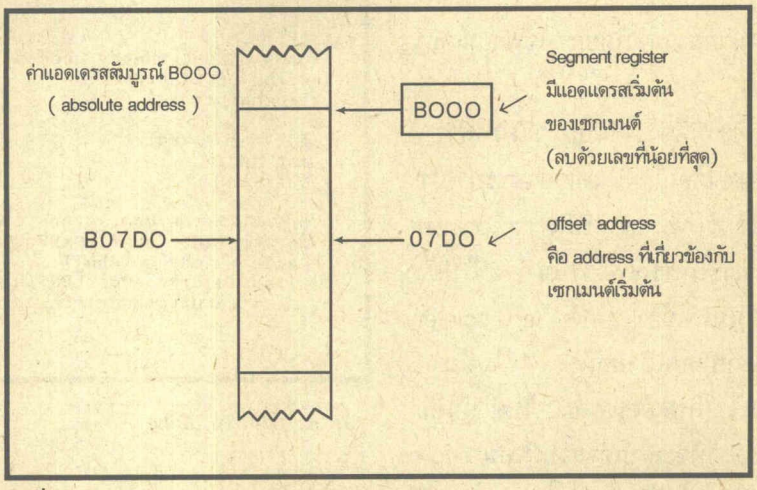
ตามปกติ (อย่างน้อยในรูปจำลองของหน่วยความจำขนาดเล็ก) ข้อมูลทั้งหมดในโปรแกรมภาษาซีจะอยู่ในหนึ่งเซกเมนต์ ดังนั้นพอยน์เตอร์ที่มี 2 ไบต์จะทำงานได้อย่างสมบูรณ์ในทุกสถานการณ์ ส่วนการเข้าถึงแอดเดรสภายนอกเซกเมนต์จะใช้แผนการที่ต่างกันออกไป

ภายใน 80x86 จะคุมสถานการณ์ด้วยเซตของรีจิสเตอร์ชุดพิเศษเรียกว่า segment register แอดเดรสที่อยู่นอกเซกเมนต์จะถูกสร้างเป็นรูปเป็นร่างขึ้นโดยการนำเอาแอดเดรสเริ่มต้นของเซกเมนต์ที่เรียกว่า segment address กับแอดเดรสภายในเซกเมนต์ที่เรียกว่า offset segment มารวมกัน แอดเดรสเริ่มต้นของเซกเมนต์จะถูกใส่ไว้ใน segment register

ดังตัวอย่าง ลองเอาแอดเดรสที่อยู่ในหน่วยความจำของจอโมโนโครมมาครึ่งหนึ่ง นั่นคือตัวอักษรตัวที่ 1,000 ซึ่งจะเป็น 2,000 ไบต์ (ในเลขฐาน 10) หรือ 7do ในเลขฐานสิบหก

เซกเมนต์มีหน่วยความจำโมโนโครมที่ถูกสมมติให้มีค่าเริ่มต้นตรงกับที่หน่วยความจำเริ่มด้วยเช่นเดียวกัน คือที่แอดเดรส B000

Segment register จะเก็บจำนวนบนสุด 4 ตัวแรก หรือ B000 ซึ่งจะแสดงให้ดูดังรูปที่ 2



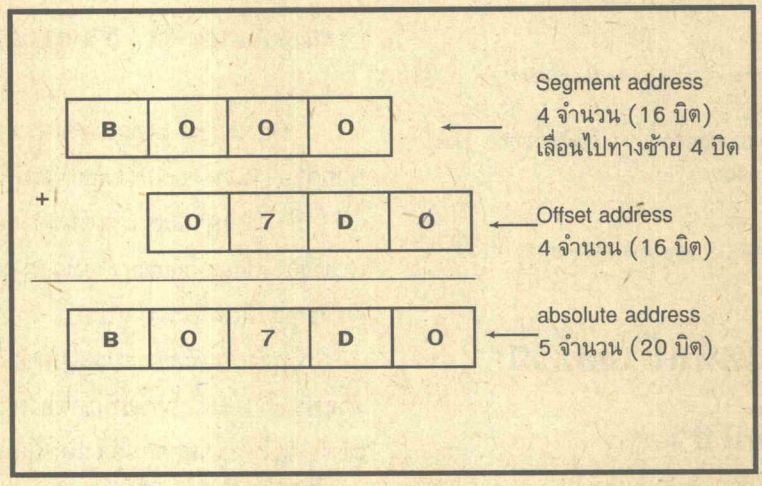
รูปที่ 2 แสดงส่วน Segment register และ Video Memory

การหาค่าแอดเดรส

แอดเดรสที่อยู่ใน segment register ถูกรวมกันเข้ากับ offset address การเลือกที่จะรวมเกิดขึ้นด้วยวิธีประหลาด กล่าวคือ ส่วนที่บรรจุอยู่ใน segment register ถูกเลื่อนไปทางซ้าย 4 บิต แล้วบวกเข้ากับ offset address จากรูปที่ 3 แสดงถึงกรณีของแอดเดรสตอนส่วนกลางของหน่วยความจำที่แสดงให้ปรากฏ

จากรูปที่ 3 segment address, B000 ถูกเลื่อนไปทางซ้าย 4 บิต แล้วบวกกับ offset address, O7DO ผลลัพธ์คือแอดเดรสความจำจริงหรือสมบูรณ์

ส่วนในไมโครโปรเซสเซอร์ absolute address หาได้จากการเลื่อน segment address ไปทางซ้าย 4 บิต แล้วบวกกับ offset address



รูปที่ 3 การเชื่อม segment และ offset address

การใช้ segment-offset address ใน C

ใน C แอดเดรสที่อยู่นอกเซกเมนต์ข้อมูลปกติของโปรแกรมก็นำเอาประโยชน์ของการผสมกันเป็น segment-plus-offset อย่างไรก็ตาม address ที่ใช้เป็นตัวแทนนั้นมีความแตกต่างกัน แทนที่จะใช้ absolute address ใน C ต้องการ 32 บิต (4 ไบต์) ที่ใช้เป็นตัวแทน จะประกอบด้วย 4 จำนวนสำหรับ segment address ตามด้วย 4 จำนวนสำหรับ offset address ดังนั้นใน C absolute address BO7DO จะเขียนเป็น OxB00007DO (B000 ตามด้วย O7DO)

ใน C 32-bit pointer จะถูกคำนวณโดยการเลื่อน segment address ไปทางซ้าย 16 บิต แล้วบวกด้วย offset address นั่นคือถ้าพอยน์เตอร์ปกติจะไม่สามารถเก็บ 32-bit address แล้วเราจะใช้ประโยชน์มันได้อย่างไร นั่นคือปัญหา เรามีวิธีการแก้ปัญหาได้หลายแบบคือใช้แบบจำลองหน่วยความจำขนาดใหญ่ (ซึ่งจะกล่าวถึงแบบจำลองหน่วยความจำในบทความต่อ ๆ ไป) ในแบบจำลองหน่วยความขนาดใหญ่พอยน์เตอร์ทุกตัวจะกินเนื้อที่ 4 ไบต์ ไม่ใช่ 2 ไบต์ ดังนั้นความไม่ดีของหน่วยความจำขนาดใหญ่คือ ข้อมูลทุกตัวที่จะถูกอ้างถึงจะต้องใช้พอยน์เตอร์ขนาด 4 ไบต์ด้วย ซึ่งทำให้ไม่มีประสิทธิภาพ เพราะกินเวลานานที่จะ

เข้าถึงหน่วยความจำโดยการใช้พอยน์เตอร์ 4 ไบต์

ทางที่ดีควรใช้แบบจำลองหน่วยความจำขนาดเล็ก แต่เวลาประกาศใช้พอยน์เตอร์ให้ประกาศเป็น far pointer โดยเฉพาะในกรณีที่จำเป็นที่จะชี้ไปยังส่วนเซกเมนต์ข้อมูลปกติ far pointer เป็นตัวเก็บแอดเดรสขนาด 4 ไบต์มากกว่าที่จะใช้เก็บแอดเดรส 2 ไบต์ นั่นคือมันสามารถที่จะเก็บเอาส่วนที่เป็น segment offset ได้ด้วย

การประกาศ far pointer ก็เพียงแต่เพิ่มคีย์เวิร์ดก่อนหน้าชื่อตัวแปร เช่น

```
int far *ptr; /* declares far pointer*/
---------
ptr =0XB0000000; /*segment B000 plus offset 0000*/
----------
*(ptr)=ch; /*new this will work*/
```

การเติมหน้าจอด้วยตัวอักษร

เราจะลองใช้วิธีการ far pointer เติมหน้าจอด้วยตัวอักษรหนึ่งตัว 2,000 ครั้ง แล้วดูว่ามันจะเร็วขนาดไหนเมื่อเปรียบเทียบกับการใช้รูทีนในไลบรารี putch() และยังจะเปลี่ยนตัวอักษรได้อีกด้วยเมื่อกดคีย์ แต่ละครั้งที่ผู้ใช้กดคีย์ใหม่จอภาพก็จะถูกเติมจนเต็มจอ ถ้าผู้ใช้กด [ENTER] โปรแกรมก็จะหยุดการทำงาน

จำไว้นิดว่าถ้าคุณใช้จอสีที่แสดงข้อความ ควรใช้ค่าคงที่ OxB8000000 ถ้าใช้จอโมโนโครมให้ใช้ OxB0000000

ข้อสังเกตจากโปรแกรม เราใช้ for ลูป โดยเริ่มตั้งแต่ 0 ถึง 2,000 เติมแอดเดรสความจำทุกที่ในความจำโมโนโครมด้วยประโยค

```
/*dfill.c*/
/*uses direct memory access to fill screen*/
#define LENGTH 2000
 void main(void)
 {
  int far *farptr;
  int addr;
  char ch;

  printf("Type character to start,type again to change");
  farptr=(int far *) 0xB8000000;        /*B000000 for monochrome*/
  while ( (ch= getche() ) != '\r')      /*terminate on [Enter]*/
    for (addr=0;addr<LENGTH;addr++)
      *(farptr +addr)=ch |0x0700;
 }
```

รูปที่ 4 โปรแกรม dfill.c

```
**farptr+addr)=ch : 0 x 0700;
```

โดยจะอ้างถึงแต่ละแอดเดรส แล้วบอกด้วยจำนวน addr ซึ่ง addr จะมีค่าตั้งแต่ 0 ถึง 1999 และค่าเริ่มต้นของแอดเดรสของ video memory ซึ่งเรากำหนดค่า farptr ก่อนที่จะเข้าลูป

ทางขวามือของประโยคข้างต้น ตัวแปร ch เป็นตัวอักษรที่เราต้องการให้เข้าไปอยู่ในหน่วยความจำ โดยป้อนเข้าทางคีย์บอร์ด ค่าคงที่ OxO700 เป็นลักษณะเฉพาะของไบต์ เลื่อนไปทางซ้าย 1 ไบต์ และจะวางอยู่ทางซ้ายของปริมาณ 2 ไบต์ ค่าคงที่ OxO7 เป็นแอตทริบิวต์ปกติที่ไม่ได้สร้างการกะพริบ ตัวหนา แต่สร้างตัวอักษรสีขาวบนพื้นสีดำ ซึ่งเราจะพบต่อไป

การหมุนวนลูป 2,000 ครั้งเพื่อใส่จำนวนเต็ม 2,000 จำนวนในหน่วยความจำไบต์ที่สูงของจำนวนเต็มเป็นการขยายรหัสอักษร และไบต์ที่ต่ำแสดงถึงแอตทริบิวต์

ประโยคที่จะอธิบายต่อไปคือ

```
farptr=(int far*) 0xB8000000;
```

นิพจน์ในวงเล็บคืออะไร ทำไมจึงจำเป็นต้องมีด้วย ปัญหาคือค่าคงที่ 0xB8000000 และตัวแปร farptr เป็นข้อมูลต่างชนิดกัน ค่าคงที่ดูเหมือนจะเป็นจำนวนเต็มชนิด long ขณะที่ farptr เป็น far pointer ที่ชี้ไปยัง int

แอตทริบิวต์ไบต์ต้องการคำอธิบายเพิ่มเติม ซึ่งจะอธิบายตอนหลัง ลองเปลี่ยนโปรแกรม dfill.c เล็กน้อยเป็น dfill2.C

โปรแกรมที่จะกล่าวถึงต่อไป เราจะใช้ประโยชน์ของฟังก์ชันซึ่งรับอินพุตเป็นแถวและหลักที่ทำให้ตัวอักษรแสดงออกจอภาพ ณ ตำแหน่งที่ต้องการโดยอาศัยการเข้าถึงหน่วยความจำโดยตรง บรรทัดสุดท้ายของโปรแกรม ตัวแปร row และ col แทนแถวและหลักที่ตัวอักษรจะถูกแทรกเข้าไป แอดเดรสของหน่วยความจำที่สมนัยกันจะถูกค้นพบโดยการคูณแถวด้วยจำนวนคอลัมน์ในหนึ่งแถว แล้วบวกด้วยผลลัพธ์และจำนวนคอลัมน์ไปยังแอดเดรสเริ่มต้นของหน่วยความจำโมโนโครมใน farptr ซึ่งเราทำให้จอภาพเป็น 2 มิติแบบแถวและคอลัมน์ (มากกว่าจะใช้เพียงมิติเดียวของแอดเดรสในหน่วยความจำ) ทำได้โดยลูป 2 ลูปซ้อนกัน

ซอฟต์แวร์

การเขียนกราฟิกอย่างมีระบบ

ตอน การเข้าถึงหน่วยความจำโดยตรง (จบ)

```
          /*wpro2.c*/
/*rudimentary word-processing program*/
/*with delete,insert and underlining */
#include <dos.h>          /*for REGS definition*/
#define COMAX 80          /*max number of column*/
#define R_ARRO 77         /*right arrow*/
#define L_ARRO 75         /*left arrow*/
#define BK_SPC 8          /*backspace*/
#define ALT_U 22          /*[Alt] and [u] key combination*/
#define VIDEO 0x10        /*video ROM BIOS service*/
#define NORM 0x07         /*normal attribute*/
#define UNDR 0x01         /*underline attribute */
void cursor(void);        /*prototype*/
void insert(char,char);
void delete(void);
void clear(void);
int col=0;                /*cursor position*/
int length=0;             /*length of phrase*/
int far *farptr;          /*pointer to vedio memory*/

void main(void)
{
 char ch,attr=NORM;
 farptr=(int far *) 0xB8000000; /*( B0000000 for mono) */
 clear();                       /*clear screen*/
 cursor();                      /*position cursor*/
 while( (ch=getch() ) !='\r')   /*exit on [Enter]*/
   {
    if(ch==0)                 /*if char is 0 */
     {
      ch=getch();             /*read extended code*/
      switch(ch)
       {
        case R_ARRO : if(col<length) ++col;break;
        case L_ARRO : if(col>0)      --col; break;
        case ALT_U  : attr=(attr==NORM) ? UNDR :NORM;
       }
     }
     else
      switch(ch)
       {
        case BK_SPC: if (length >0) delete();break;
        default: if(length<COMAX) insert(ch,attr);
       }
       cursor();
     }
    }
    /*cursor()*/
    /*move cursor to row=0,col */
    void cursor(void)
    {
     union REGS regs;
     regs.h.ah=2;             /*'set cursor pos' service*/
     regs.h.dl=col;           /*column varies*/
     regs.h.dh=0;             /*always top row*/
     regs.h.bh=0;             /*page zero*/
     int86(VIDEO,&regs,&regs);  /*call video interrupt*/
    }
    /*insert() */
    /*inserts character at cursor position*/
    void insert (char ch,char attr)
    {
     int j;
      for (j=length;j>col; j--)     /*shift chars left*/
        *(farptr +j)= *(farptr+j-1);/*  to make room*/
      *(farptr+col)=ch | attr<<8;    /*insert char */
      ++length;                     /*increment count*/
      ++col;                        /*move cursor right*/
     }
     /*delete()*/
     /* clears screen by inserting 0 at every location */
     void delete(void)
     {
      int j;
      for(j=col;j<=length;j++)           /*shift chars right*/
        *(farptr +j-1)=*(farptr+j);
        --length;                        /*decrement count*/
        --col;                           /*move cursor left*/
      }
       /*clear screen by inserting 0 at every location*/
       void clear(void)
       {
        int j;

        for(j=0;j<2000;j++)
          *(farptr +j)=0x0700;
        }
```

นอกจากจะมีการขีดเส้นใต้ เรายังเพิ่มการแทรกตัวอักษร การลบตัวอักษร การแทรกตัวอักษรผู้ใช้ก็เลื่อนเคอร์เซอร์ไปยังจุดที่ต้องการจะพิมพ์ ตัวอักษรที่อยู่หลังเคอร์เซอร์ก็จะเลื่อนไปทางขวา 1 ตัวอักษร เพื่อทำให้เกิดที่ว่าง การลบตัวอักษรผู้ใช้กดคีย์ backspace ซึ่งเป็นการลบตัวอักษรที่อยู่ทางซ้ายของเคอร์เซอร์ ตัวอักษรที่อยู่ทางขวาก็จะเลื่อนมาทางซ้ายเพื่อไม่ให้เกิดที่ว่างดังโปรแกรม wpro2.c

เมื่อกด [alt][u] (คือการกด Alt ค้างไว้แล้วกด u)

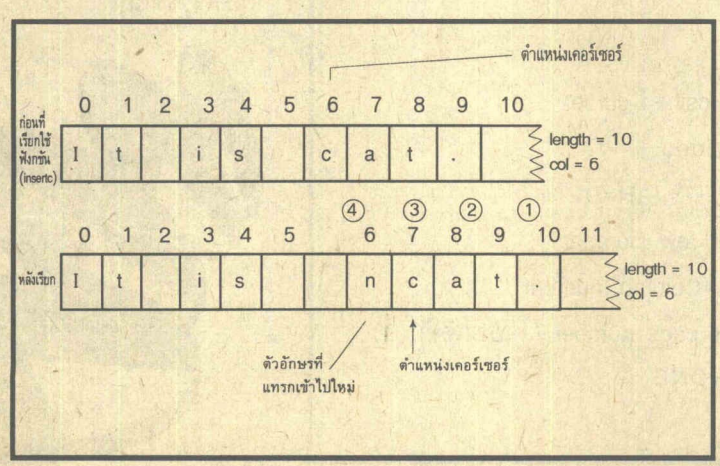
นิพจน์เงื่อนไข (ที่อยู่ตอนท้ายของโครงสร้าง switch)

```
attr=(attr == NORM)?UNDR : NORM;
```

ช่วยในการเปลี่ยนกลับไปกลับมาระหว่าง UNDR และ NORM ซึ่งใช้ 07 (ปกติ) และ 01 (ขีดเส้นใต้) เป็นแอตทริบิวต์ไบต์

ฟังก์ชัน insert() เป็นการใช้ for ลูป ซึ่งลูปนี้ใช้เข้าถึงหน่วยความจำโดยตรง และ far pointer ช่วยเลื่อนตัวอักษรทั้งหมดที่อยู่ทางขวาของเคอร์เซอร์ไปทางขวา 1 ตัวอักษร กระบวนการเลื่อนนี้ต้องเริ่มทางขวาสุดของวลีที่มีอยู่ ถ้าการเลื่อนเริ่มขึ้นทางซ้าย ตัวอักษรซ้ายสุดจะถูกเขียนทับตัวอักษรที่อยู่ถัดไปก่อนที่จะถูกเลื่อน เมื่อตัวอักษรถูกเลื่อนไปหมดแล้ว ตัวอักษรที่เกิดจากผู้ใช้แทรกเข้าไปจะอยู่ตรงตำแหน่งเคอร์เซอร์ (รูปที่ 15)

รูปที่ 15



การลบอักษรก็ทำทำนองเดียวกัน อักษรทุกตัวที่อยู่ทางขวาของเคอร์เซอร์จะถูกเลื่อนไปทางซ้าย 1 ตัวอักษร อักษรที่อยู่ทางซ้ายสุดจะแทนที่อักษรถูกลบทิ้งไป ในกรณีที่การเลื่อนเริ่มขึ้นซ้ายมือสุด

ก็หวังว่าโปรแกรมเวิร์ดโปรเซสซิ่งคงจะให้แนวคิดไม่มากก็น้อยเกี่ยวกับความเป็นไปทางการเข้าถึงความจำโดยตรง

รูปที่ 7
โปรแกรม wpro1.c

```
   /* wpro1.c*/
/* rudimentary word-processing program */
/* allows typing,cursor movement,and typeover*/
# include "dos.h"            /*for  REGS definition */
#define COMAX 80             /*max number of columns*/
#define R_ARRO 77            /*right arrow */
#define L_ARRO 75           /*left arrow*/
#define VIDEO 0x10           /*video ROM BIOS services*/
#define CTRL_C  '\x03'       /* [ctrl] [c]*/
int col=0;                   /*cursor position */
int far *farptr;             /*video memory pointer*/
union REGS regs;             /*for ROM BIOS calls*/
void cursor(void);           /*prototypes*/
void insert(char);
void clear(void);

void main(void)
{
 char ch;
 farptr=(int far *)0xB8000000;  /*start of screen memory*/
 clear();                       /*clear screen*/
 cursor();                      /*position cursor*/
 while( (ch=getch() ) !='\r')   /*exit on [Enter]*/
   {
   if(ch== 0)                   /* if ch  is 0*/
     {
      ch=getch();               /*read exteneded code */
      switch(ch)
        {
        case R_ARRO: if(col<COMAX) ++ col; break;
        case L_ARRO: if(col>0)     --col; break;
        }
      }
      else                      /* not exteneded code*/
      if (col<COMAX)            /*print character at col*/
        insert(ch);
       cursor();                 /*reset cursor*/
      }
     }
    /*cursor() */
    /*move cursor to row=0,col*/
    void cursor(void)
    {
      regs.h.ah=2;              /*'set cursor position' service*/
      regs.h.dl=col;            /*col varies*/
      regs.h.dh=0;              /*always top row*/
      regs.h.bh=0;              /*page zero*/
      int86(VIDEO,&regs,&regs);  /*call video interrupt */
    }
    /*insert()*/
    /*inserts character at cursor position */
    void insert(char ch)
    {
     int j;
     *(farptr +col)=ch |0x700;     /*insert char */
     ++col;                        /*move cursor right*/
    }
    /*clear() */
    /*clears screen using direct memory access*/
    void clear(void)
    {
     int j;
     for(j=0;j<2000; j++)          /*fill screen memory*/
       *(farptr+ j)=0x0700;        /*with 0's (attr=07) */
     }
```

รูปที่ 8

ROM BIOS routine : Position cursor

Interrupt 10 hex : Video

Input registers : AH=02

DH=row number

DL=Column number

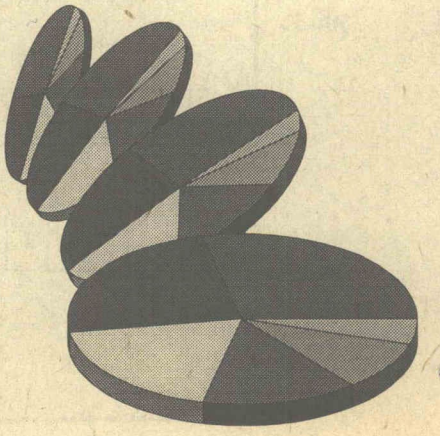
BH=page number (โดยปกติคือ 0)

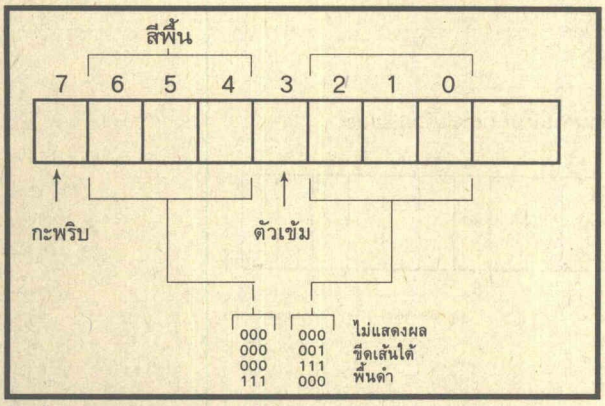
OUTPUT registers : NONE

แอตทริบิวต์ไบต์
(The Attribute byte)

เราได้เรียนรู้มาก่อนแล้วว่า โลเคชันในหน่วยความจำกราฟิกนั้นสมนัยกับตำแหน่งของตัวอักษรเดี่ยวที่อยู่บนจอภาพ ซึ่งประกอบด้วย 2 ไบต์ ไบต์หนึ่งจะเก็บรหัสส่วนขยายของตัวอักษร อีกไบต์หนึ่งจะเก็บลักษณะเฉพาะของมัน จากรูปที่ 9 แสดงวิธีการแบ่งแอตทริบิวต์ไบต์ออกเป็นส่วน ๆ มีอยู่สองส่วนที่ประกอบด้วยบิตเดียวคือ Bit3 จะควบคุมความเข้ม (intensity) Bit 7 จะดูแลเรื่องการกะพริบ ถ้ามีบิตใดบิตหนึ่งเป็น 1 ก็จะทำให้แอตทริบิวต์ที่สมนัยกัน (กะพริบหรือตัวเข้ม) จะปรากฏขึ้นเมื่อให้บิตนั้นมีค่าเป็น 0 แอตทริบิวต์นั้นก็จะไม่ปรากฏว่ากะพริบหรือมีตัวเข้มเกิดขึ้น

สองส่วนที่เหลือของแอตทริบิวต์ไบต์แต่ละส่วนประกอบด้วย 3 บิต บิต 0, 1, และ 2 ช่วยทำสี foreground ขณะที่บิต 4, 5, 6 สร้างสีพื้น (background color) สำหรับจอโมโนโครมก็ยึดรูปแบบเดียวกัน โดยมีให้เลือก 3 แบบ ดำ ขาว (สีเขียวหรืออำพันขึ้นอยู่กับจอภาพของคุณ) และการขีดเส้น





รูปที่ 9 แสดงการแบ่งแอตทริบิวต์ออกเป็นส่วน ๆ

```
/*dfill3.c*/
/*fills screen using blinking attribute*/
#define ROMAX 25
#define COMAX 80
void main(void)
{
 int far *farptr;
 int col,row;
 char ch;

 printf("Type character to start,type again to change");
 farptr=(int far *) 0xB8000000; /*B8000000 for monochrome*/
 while ((ch=getche()) !='\r')    /*terminate on [Enter]*/
  {
    for(row=0;row<ROMAX; row++)
      for(col=0;col<COMAX;col++)
       *(farptr+row*COMAX+col)=ch |0x8700;
       }
 }
```

รูปที่ 10 โปรแกรม dfill3.c

```
/*attrfill.c*/
/*uses direct memory access to fill screen*/
#define ROMAX 25
#define COMAX 80
void fill(char,char);   /*prototype */

void main(void)
{
  char ch,attr;

  printf("Type 'n' for normal,\n");
  printf("       'u' for underlined,\n");
  printf("       'i' for intensified,\n");
  printf("       'b' for blinking,\n");
  printf("       'r' for reverse video,\n");
  while((ch=getch()) !='\r')
    {
     switch(ch)
      {
       case 'n':
        attr=0x07;       /*set to normal*/
        break;           /*0000 0111*/
       case 'u':
        attr=attr & 0x88; /*set to underline*/
        attr=attr |0x01;  /*x000 x001*/
        break;
       case  'i':
       attr=attr^ 0x08;   /*toggle intensified*/
       break;             /*xxxx Txxx*/
       case 'b':
        attr=attr ^ 0x80; /*toggle blinking*/
        break;            /*Txxx xxxx*/
       case 'r':
         attr= attr & 0x88; /*set to reverse */
         attr=attr |0x70; /*x111 x000*/
         break;
        }
       fill(ch,attr);
       }
      }
      /*fill() */
      /*fills screen with character 'ch',attribute 'attr' */
      void fill(char ch,char attr)
       {
        int far *farptr;
        int col,row;

        farptr=(int far *) 0xB8000000; /*50000000 for monochrome*/
        for(row=0;row<ROMAX;row++)
         for(col=0;col<COMAX;col++)
            *(farptr +row*COMAX+col)= ch |attr<<8;
         }
```

รูปที่ 11 โปรแกรม attrfill.c

ลองดัดแปลงโปรแกรม dfill2.c เป็นโปรแกรม dfill3.c ที่ทำให้ตัวอักษรทุกตัวกะพริบ แต่ไม่สามารถทำให้กลับเป็นปกติได้เว้นแต่จะเข้าไปที่ดอสแล้วใช้คำสั่ง cls (เพื่อลบจอภาพ) โปรแกรมใช้แอตทริบิวต์ 07 ซึ่งก็คือ 00000111 (ฐาน 2) นั่นคือ 3 บิตที่เป็น foreground กำหนดให้เป็น 1 นอกนั้นเป็น 0 ถ้าต้องการให้กะพริบจะต้องทำให้บิต 7 มีค่าเป็น 1 นั่นคือ 10000111 ซึ่งตรงกับ 87 ฐาน 16

บิตฟิลด์ (Bit Fields)

โปรแกรม attrfill.c เป็นการเข้าถึงแต่ละบิตและกลุ่มของบิตโดยอาศัยบิตไวซ์โอเปอเรเตอร์ เราสามารถเข้าถึงบิตด้วยวิธีการที่แตกต่างจากที่กล่าวมาแล้วด้วยวิธีการใช้บิตฟิลด์

บิตฟิลด์เป็นโครงสร้างชนิดพิเศษ สมาชิกของโครงสร้างคือจำนวนบิตจำนวนบิตของแต่ละสมาชิกจะอยู่หลังเครื่องหมาย : และเป็นจำนวนที่ประกาศไว้ด้วย unsigned int เช่น การประกาศบิตฟิลด์ (bit-field) รูปที่ 12

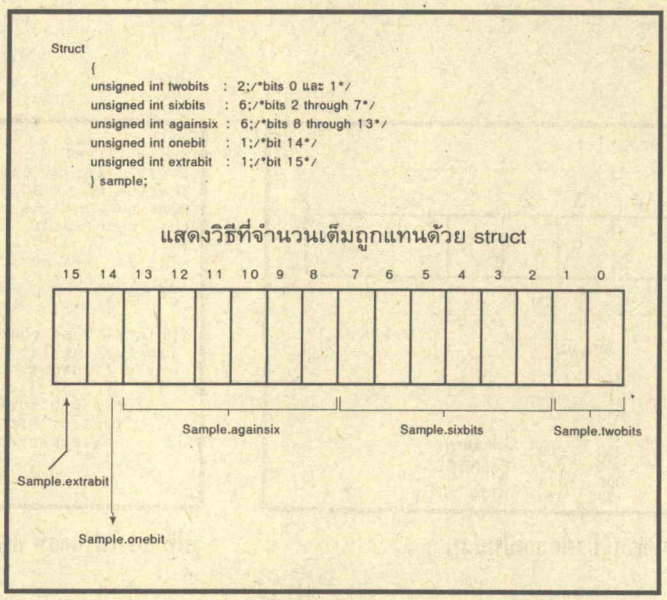
การเข้าถึงสมาชิกของฟิลด์ก็เหมือนกับการเข้าถึงสมาชิกของ struct ในรูปแบบอื่นด้วยการใช้โอเปอเรเตอร์ ที่ช่วยเชื่อมชื่อของตัวแปรของ struct กับชื่อของสมาชิกของ struct เช่น

```
sample.twobits=3;
sample.sixbits=63;
sample.onebit=1;
```

ข้อสังเกต เราไม่สามารถจะให้ค่าแก่สมาชิกที่เกินความจุของมัน

ฟิลด์ 1 บิต จะมีได้เฉพาะ 0 กับ 1 เท่านั้น

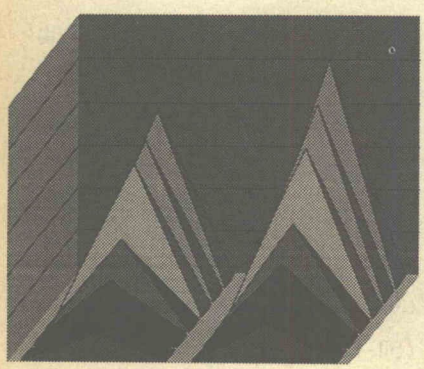
ฟิลด์ 6 บิต จะมีค่าได้ตั้งแต่ 0 ถึง 63 attrfill2.c เป็นเวอร์ชันใหม่ของ attrfill .c ที่ใช้บิตฟิลด์เข้าถึงส่วนต่าง ๆ ของแอตทริบิวต์ไบต์ เนื่องจากแอตทริบิวต์ต้องถูกเข้าถึงทั้งบิตฟิลด์และจำนวนเต็ม จึงต้องใช้ UNION เข้าช่วย



รูปที่ 12
บิตฟิลด์

ลองมาดูเวิร์ดโปรเซสซิงอีกครั้ง

มาถึงตรงนี้เราได้ทราบเรื่องของแอตทริบิวต์ เราจึงขยายรากฐานของโปรแกรมเวิร์ดโปรเซสซิงออกไป เช่น มีการขีดเส้นใต้โดยการกด [alt][u] ถ้าต้องการยกเลิกก็กด [alt] [u] อีกครั้ง (ถ้าเป็นการเล่นสี การใช้ [alt][u] ก็คือการเปลี่ยนสี)



```
/*atrrf2.c*/
/*uses direct memory access to fill screen*/
#define ROMAX 25
#define COMAX 80
void fill (char,unsigned int);
void main(void)
{
 struct bits            /*bit fields*/
  {
   unsigned int foregnd :3;      /*bits 0,1,2*/
   unsigned int intense :1;      /*bit 3*/
   unsigned int backgnd :3;      /*bits 4,5,6*/
   unsigned int blinker :1;      /*bit 7*/
    }b1;                         /*define struct variable*/
 union
   {                             /*declare union*/
   unsigned int attr;            /*unsigned int and*/
   struct bits b1;               /*bitfield*/
   }u1;                          /*define union variable*/
  char ch;
  printf("Type 'n' for normal,\n");
  printf("      'u' for underlined,\n");
  printf("       'i' for intensified,\n");
  printf("       'b' for blinking,\n");
  printf("      'r' for reverse video,\n");
  while(( ch=getch()) !='\r') /*Terminate on [Enter] */
    {
     switch(ch)
      {
       case 'n':    /*set to normal */
         u1.b1.foregnd=7;
         u1.b1.backgnd=0;
         break;
        case 'u':   /*set to underlined*/
          u1.b1.foregnd=1;
          break;
          case 'i': /*toggle intensity*/
           u1.b1.intense=(u1.b1.intense==1) ? 0:1;
           break;
           case 'r':    /*set to reverse */
            u1.b1.foregnd=0;
            u1.b1.backgnd=7;
            break;
           }
           fill(ch,u1.attr);
           }
          }
         /*fill()*/
         /*fills screen with character 'ch',attribute 'attr' */
         void fill(char ch,unsigned int attr)
         {
          int far *farptr;
          int col,row;

          farptr=(int far *) 0xB8000000; /*B0000000 for mono*/
          for(row=0; row<ROMAX; row++)
```

รูปที่ 13 โปรแกรม attrfill2.c

```
/*dfill2.c*/
/*uses direct memory access to fill screen*/
#define ROMAX 25
#define COMAX 80
void main(void)
{
 int far *farptr;
 int col,row;
 char ch;

 printf("Type  character to start,type again to change");
 farptr=(int far *)0xB8000000;  /*B000000 for monochrome*/
 while ( (ch=getche() ) !='\r')
   for (row =0;row<ROMAX;row++)
     for(col=0;col<COMAX;col++)
      *(farptr+row*COMAX+col)= (int)ch | 0x0700;
  }
```

รูปที่ 5 โปรแกรม dfill2.c

```
   /* cfill.c*/
/* uses putch() to fill screen*/
#define ROMAX 25
#define COMAX 80
#define CLEAR "\x1B[2J"
void main(void)
{
 int col ,row;
 char ch;

 printf("Type character to start ,type again to change");
 while ( (ch=getche()) !='\r')
  {
    printf(CLEAR);
    for(row=0;row<ROMAX;row++)
     for(col=0; col<COMAX; col++)
      printf("%c",ch);
  }
 }
```

รูปที่ 6 โปรแกรม cfill.c

การเปรียบเทียบความเร็ว

เพื่อที่จะได้แนวคิดในการใช้ประโยชน์ในแง่ความเร็วที่อาศัยการเข้าถึงหน่วยความจำโดยตรงด้วย far pointer เราจะเขียน dfill2.c ใหม่ด้วยฟังก์ชัน putch() ที่เป็นฟังก์ชันอยู่ในไลบรารี โปรแกรม cfill.c จะเติมอักษร 1 ตัวลงจอภาพจนเต็ม แต่จะใช้เวลามากกว่า dfill2.c ถึง 20 เท่า

เวิร์ดโปรเซสเซอร์บรรทัดเดียว

เพื่อจะชี้ให้เห็นถึงการใช้ farpointers ที่นอกเหนือจากการเติมอักษร 1 ตัวเต็มจอภาพแล้วจะลองสร้างโปรแกรมเวิร์ดโปรเซสซิงที่ทำงานอยู่บนบรรทัดเดียว วิธีการทำจะมีสองขั้นตอน ขั้นแรกจะให้ผู้ใช้ป้อนตัวอักษรไปหนึ่งบรรทัดแล้วเคลื่อนย้ายเคอร์เซอร์ไปข้างหน้าหรือข้างหลังตามเส้นบรรทัดนั้น ตัวอักษรจะถูกลบด้วยการเลื่อนเคอร์เซอร์ไปทับตัวอักษรนั้น หลังจากที่ได้เรียนลักษณะจำเพาะของแต่ละตัวอักษรแล้วจึงค่อยเพิ่มการบวก การลบ และการขีดเส้นใต้ดังโปรแกรม wpro1.c

โปรแกรมนี้คงจะเป็นที่คุ้นเคยสำหรับคุณ เพราะเป็นโปรแกรมที่คุณใช้พิมพ์รายงาน โดยในขั้นแรกจะทำการลบหน้าจอด้วยการใช้รูทีนที่เติม 0 ลงในหน่วยความจำจอภาพ (ซึ่งก็เหมือนกับที่ผมได้นำเสนอไปแล้วด้วยการเติมอักษรลงจอภาพด้วยโปรแกรม dfill.c) หลังจากนั้นเคอร์เซอร์จะเลื่อนไปอยู่ต้นบรรทัดด้วยวิธีกำหนดตำแหน่งของเคอร์เซอร์จากการบริการของวิดีโอรอมไบออส (Video ROM BIOS SERVICE) (รูปที่ 8)

การกดคีย์เคอร์เซอร์จะเป็นการเพิ่มหรือลดค่าตัวแปร col ซึ่งชี้ถึงตำแหน่งที่เคอร์เซอร์อยู่ การเรียกฟังก์ชัน cursor() จะทำการเลื่อนเคอร์เซอร์ให้อยู่ที่คอลัมน์นี้

ถ้าตัวอักษรที่ถูกป้อนเข้าไปไม่ใช่อักษรขยาย (มีชื่อว่าคีย์เคอร์เซอร์) มันจะถูกแทรกเข้าไปที่หน้าจอตรงกับคอลัมน์ col ด้วยการใช้รูทีน insert() ซึ่งเราก็คุ้นจากการที่ได้ถูกใช้ far pointer ตัวแปร col จะมีเพิ่มขึ้นเมื่อมีตัวอักษรมาแทรก ดังนั้นตัวอักษรตัวต่อไปจะถูกป้อนเข้าทางขวาของอักษรที่มีอยู่เดิม